VeCycle: Recycling Checkpoints for Faster Migrations

Thomas Knauth, Christof Fetzer
Technische Universität Dresden
Introduction

• Lots of papers about VM migration
• Not that widely used in practice
  • migration consumes lots of resources
  • unavailability of VM during migration
• Problem: must shuffle gigabytes of data across the network [1]

[1] V. Soundararajan et al., The Impact of Management Operations on the Virtualized Datacenter, ISCA, 2010
Motivation: Energy Proportional Computing

### Table I

<table>
<thead>
<tr>
<th>Name</th>
<th>Function</th>
<th>Ref.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Native</td>
<td>An unchanged version of the application. However assertions do not trigger an abort. Instead, violated assertions are logged and ignored.</td>
<td></td>
</tr>
<tr>
<td>Assertion</td>
<td>Violated assertions cause an abort of the application.</td>
<td></td>
</tr>
<tr>
<td>OverAllocate</td>
<td>Pads stack and heap allocations to tolerate present buffer overflows. Stack allocations are padded on a per object basis, so each stack variable is padded.</td>
<td>[8, 9]</td>
</tr>
<tr>
<td>NullWrite</td>
<td>Ignores writes to address 0x0.</td>
<td>[14]</td>
</tr>
<tr>
<td>NullRead</td>
<td>Return 0 if address 0x0 is loaded.</td>
<td>[14]</td>
</tr>
<tr>
<td>NullDeref</td>
<td>Combines NullWrite and NullRead.</td>
<td></td>
</tr>
<tr>
<td>SWIFT</td>
<td>SWIFT duplicates all instructions and registers apart from memory accesses and control flow instructions to detect transient hardware errors. Since the original implementation of SWIFT was not available, we reimplemented SWIFT.</td>
<td>[15]</td>
</tr>
<tr>
<td>SWIFTCFC</td>
<td>SWIFT alone does not detect control flow errors. SWIFTCFC adds control flow checking to SWIFT.</td>
<td>[4]</td>
</tr>
</tbody>
</table>

A large number of software diversification methods are available (e.g. [7, 4, 8, 9, 10, 11, 12]). In principle, Aaron can use any of those. We implemented multiple software diversification methods (Table I). The goal is neither to present new diversification methods nor to extend knowledge about existing methods. Instead, we use them to prove the applicability and the overall soundness of Aaron. Aaron can be easily extended with new diversification methods if desired.

The diversification itself is achieved using the LLVM compiler infrastructure [13]. LLVM translates the source code of the application into an intermediate representation (IR). For each diversification method, the intermediate representation is copied and augmented with runtime checks (Figure 2). All functions are renamed and calls and references are adjusted accordingly. Global variables are shared among variants. Finally, all variants and the global variables are linked into a single file. Thus, we gain multiple versions, i.e. variants, of the application. Each variant protects against a specific set of errors. All variants together form the diversified application.

### C. Power Consumption

Computers consume a significant amount of power even while being idle or only lightly loaded. Figure 3 shows the power consumption of one of our Dell Precision R5400 machines, measured with the help of a Raritan Dominion PX-5528 power distribution unit.

If the node is not loaded at all, it already consumes 133 watts. Power consumption increases linearly and peaks at 190W. At a load of 50%, power consumption is already 88% of consumption under full load (167 watts). Only a small increase of about 14% has to be paid in order to use the remaining 50% of CPU power.

To handle sudden load surges, cluster deployments are often oversized [16]. Aaron uses those spare cycles present in deployed systems to schedule different diversified software variants to detect and tolerate runtime failures. Since Aaron uses spare cycles, we expect only a small increase in total power consumption.

### D. Scheduling Software Variants

To be able to exploit spare cycles in deployed systems without generating any user-perceivable runtime penalty, Aaron has to adapt to varying workloads extremely fast. Aaron currently relies on an important fact of applications running in cluster environments: cluster applications process server power consumption

![Graph showing server power consumption versus CPU utilization](image.png)

- **Energy consumption [watts]**
- **CPU utilization [%]**
VM Consolidation

remote access

Remote access allows users to connect to virtual machines (VMs) hosted on a consolidation server. This setup enables users to switch off desktops while keeping the VMs online, improving resource utilization and energy efficiency.

[e.g., Jettison, EuroSys 2012]
VM Bin Packing

- **Energy-proportional computing:**
  - Number of **on** servers proportional to work load

```
rack
server
server
server
server
server
VMs  VMs  VMs  VMs

more demand: switch on servers

less demand: switch off servers
```
VM Bin Packing

- energy-proportional computing:
  - number of on servers proportional to work load

- Difficult to schedule VMs such that we can indeed switch off servers
VM Bin Packing

- **energy-proportional computing:**
  - number of on servers proportional to work load

- **Difficult to schedule VMs**
  such that we can indeed switch off servers

- **Approach:**
  - schedule VMs to be able to switch off servers
  - migrate remaining VMs
Migration is used to mitigate resource hot spots.
Migration is used to mitigate resource hot spots
Observations

- VMs only migrate between a small set of servers [1]

- **spacial locality**: updates are local, constrained to a subset of the entire memory

- **temporal locality**: only a subset of the entire memory changes in any given time frame

What is the memory update rate in practice?

- upper bound on migration traffic savings
- best case: ~70%
- avg case: ~30%
- worst case: <10%

Server A, Linux, 1 GiB

<table>
<thead>
<tr>
<th>Time between snapshots [hours]</th>
<th>Snapshot similarity [0.0-1.0]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>minimum</td>
</tr>
<tr>
<td>2</td>
<td>average</td>
</tr>
<tr>
<td>4</td>
<td>maximum</td>
</tr>
</tbody>
</table>

0 0.5 1.0

10 15 20 25
VeCycle Step 1: Migrate

host A  step 1: migrate  host B
Step 2: write checkpoint at source

host A

host B

step 2: checkpoint
Step 3: combine checkpoint and current state

step 3: checkpoint-assisted migration
How to determine updates?

1. Migrate page from Host A to Host B.

VM

time

Host A

Host B
What to send?

techniques can of course be combined, i.e., CP-assisted migration with dirty tracking and de-duplication
Comparing the effectiveness of traffic reduction techniques

- baseline is a full migration
- hashes are better than dirty and/or reduplication
- hashes benefit from de-duplication too

Fraction of Baseline Traffic [0.0-1.0]
Comparison with dirty tracking plus de-dup?

- depends on the workload
- Server C benefits much more than Server A
Migration: VM Selection

minimal CP-assisted state transfer
How does it affect migration time?

- migration time increases in proportion to the percentage of updated memory
How does it affect the network traffic?

- measured outgoing network traffic at the migration source
- network traffic increases proportional to the updated memory
Summary

• migrate VMs by combining local checkpoint with remote state
• less network traffic
• migration finishes in less time
• CP-assisted migration reduces the network traffic more effectively than dirty tracking and/or de-duplication
Source Code

• Source code:
  • https://bitbucket.org/tknauth/vecycle-qemu

• Fingerprint traces:
  • http://wwwpub.zih.tu-dresden.de/~vecycle/

• Other artifacts:
  • https://bitbucket.org/tknauth/vecycle